ARPA ORDER NO.: 189-1

R-694-ARPA

May 1972

A Message Exchange for Computer Programs and Terminals

J. Carlstedt

A Report prepared for

ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA 90406

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 67 C 0141. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of Rand or of ARPA.

DOCUMENT CONTROL DATA

1. ORIGINATING ACTIVITY The Rand Corporation	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP
3. REPORT TITLE A MESSAGE EXCHANGE FOR COMPUTER PROGRAMS AND TERMINALS	
4. AUTHOR(S) (Last name, first name, initial) Carlstedt, J.	

5. REPORT DATE May 1972	6a. TOTAL NO. OF PAGES 70	6b. NO. OF REFS. -
7. CONTRACT OR GRANT NO. DAHC15 67 C 0141	8. ORIGINATOR'S REPORT NO. R-694-ARPA	
9a. AVAILABILITY/LIMITATION NOTICES DDC-A	9b. SPONSORING AGENCY Defense Advanced Research Projects Agency	

10. ABSTRACT

Documentation of the Video Graphics System Message Handler (VMH), that provides interprocess communication and message switching among Video Graphics terminal users and multiaccess and personal programs running in the connected computers --which, at Rand, includes the entire ARPA Computing Network. VMH enables terminals and programs to communicate selectively, either one-to-one or multiplexed, across the boundaries set by computer system architecture. Message privacy is preserved. Written in OS/360 Assembler Language, VMH is a set of interrelated high-priority routines available to programs via macro instructions and to terminals via communication control commands. The message path scheme is separate from the message handling, which (1) permits users to be kept up to date about message path activity, and (2) makes the control system modifiable by simply adding new control requests and status values. VMH has operated almost continuously since early 1971, supporting 32 terminals. Tested throughput averages 600-850 messages/sec on a 360/65.

11. KEY WORDS

Computer Programs
Video Graphic System
Computer Graphics
Communication Systems
Advanced Research Projects Agency
Networks

ARPA ORDER NO.: 189-1

R-694-ARPA

May 1972

A Message Exchange for Computer Programs and Terminals

J. Carlstedt

A Report prepared for

ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

i b

Published by The Rand Corporation

PREFACE

This Report, prepared under sponsorship of the Advanced Research Projects Agency, documents an invisible but essential part of the Video Graphics System, Rand's relatively low-cost multiterminal system that provides interactive graphical and keyboard access to any program running in one of the connected computers (service machines). At Rand, the 32-terminal VGS is the means of access to the facilities of the entire ARPA Experimental Computing Network, and VMH is the communications controller--the message handler that enables terminals and programs to communicate with each other and programs to communicate with other programs across the boundaries set by an IBM OS/360, while preserving message privacy.

Neither a user's nor a programmer's guide, this Report is intended primarily for those who might influence the design of systems for providing communication between or among currently executing computer programs and/or remote terminals, especially in an IBM OS/360 operating environment.

Overall descriptions of the Video Graphics System will be found in K. W. Uncapher, *The Rand Video Graphic System--An Approach to a General User-Computer Graphic Communication System*, R-753-ARPA, April 1971, and in Ellis, et al., *ARPA Network Series: I. Introduction to the ARPA Network at Rand and to the Rand Video Graphics System*, R-664-ARPA, September 1971.

SUMMARY

VMH, the Video Graphics System Message Handler, provides interprocess communication control and message switching among users of VGS terminals and application programs and multiaccess facilities in large connected computers--which, at Rand, includes the facilities of the entire ARPA Computing Network. VMH enables terminals and programs to communicate selectively with each other, either one-to-one or by multiplexing, and similarly enables programs to communicate across the boundaries set by the computer operating system architecture, while preserving message privacy. This Report gives the history, concepts, and implementation data of VMH, describes its important features and commands, and presents a final evaluation.

VMH is a part of the Video Operating System, a software component of the Video Graphics System: a relatively low-cost multiterminal system, based on closed-circuit television technology with digital/analog scan conversion, providing a full range of user/computer interaction, from remote job entry to dynamic graphics. VGS terminals are TV monitors with additional input devices. An IBM 1800 process controller routes images and other messages between the terminals and the service computers by coaxial cable. The system has two major software packages: the 1800 software and the Video Operating System, which includes control languages and VMH. The VOS software is an extension of the operating system of each System/360 service machine.

The design of VMH was strongly influenced by its OS/360 environment; much of the programming was designed to overcome limitations of the operating system or to use it with greatest efficiency. Written in OS/360 Assembler Language, VMH consists of a set of interrelated routines, some of which are supervisor calls (SVCs) and some of which are link edited together and run as asynchronous event-driven subtasks of an OS/360 job step. VMH facilities are available to executing programs via a set of macro instructions, the expansions of which include SVC instructions, and to terminal users as a set of communication control commands. A complete set of options are offered to programs with respect to message receiving and sending.

When initiated as an OS/360 job step, VMH has the highest dispatching priority of all tasks except for System tasks such as the Master Scheduler, thus ensuring that interruptions will be few and short. A basic decision was that communication control (the message path scheme) is separate from the actual message handling. This (1) permits users to be kept up to date about the message path activities of other users, and (2) enables the control system to be easily modified and adapted to new requirements simply by adding new control requests and status values. No message path may be opened unless both parties agree, meaning that users have the option of refusing unwanted information.

As actually implemented, the VMH program includes approximately 11,000 Assembler Language source statements and about 3000 macro-definition statements (not counting comments, listing controls, or macro-generated statements). The essential VMH, described in this Report, would probably be about 60 percent of that size. Design, programming, and checkout required about four man-years, of which a high proportion was spent in checkout. As tested, VMH throughput averages over 600 messages per second on a 360 model 65. Its reliability may be judged from the fact that VOS is in almost continuous operation at Rand and has supported thirty-two terminals since its installation in its present form in early 1970.

ACKNOWLEDGMENTS

VMH is the result of the pleasant cooperation of many individuals. Among the major participants, the loudest thanks must go to Dick Snow, who "took care" of many of the hardest problems: programming the interface routines with OS/360, the initiation and termination procedures, and the INPUTTER and OUTPUTTER; smoothly performing the critical tasks of library maintenance and system generation; and solving some of the most perplexing bugs. To Gerry Levitt goes much credit for the message-path scheme and the terminal agent, and for implementing the initial versions of all the service procedures.

We appreciate the help of Bill Josephs and Larry Clark, who influenced the design and facilitated the coding of the user program interface, and Richard Bisbey, who designed and implemented the terminal interface. We especially appreciate the competence and patience of Gloria Smelser, who typed countless versions of the internal documentation, and of Marian Winston, who substantially improved the style and readability of this report.

Preceding page blank

GLOSSARY OF CODES AND ACRONYMS

ARs	Attention routines
CB	Complete but blocked (of a VMH message path)
CU	Complete and unblocked (of a VMH message path)
DESTINATION_Q	Queue of messages delivered to a program but not yet accessed by it.
ECB	Event control block
EDP	In VMH, event data pointer
ID	In VMH, initiated by the destination (of a message path). A user's identification is called INTRINSIC_ID.
I_DISTRIBUTOR	Incoming message distributor
IGS	Rand's Interactive Graphics System
IMAR	Incoming message attention routine
I/O	Input/Output
IQE	Interrupt queue element
IS	Initiated by the source (of a message path)
MPAR	Message path attention routine
MFT	Multiprogramming with a fixed number of tasks
MVT	Multiprogramming with a variable number of tasks
NL	Null (of a message path)
O_DISTRIBUTOR	Outgoing message distributor
OMAR	Outgoing message attention routine
PSW	Program status word
Q	Queue
SIMAR	Specification of incoming message attention routine (for high-priority messages)
SIO	Start input/output
SOURCE_Q	A queue for each VMH user consisting of messages whose sending it has requested
SVC	Supervisor call
TCB	Task Control Block
VGS	Video Graphics System
VMH	Video Message Handler
VOS	Video Operating System, of which VMH is a part. For CONNECT/CALL VOS button of terminal and VOS-ATTENTION call, see Sec. VI.

Preceding page blank

CONTENTS

I. INTRODUCTION

This report describes a many-faceted software system, VMH, that can be viewed in different ways, depending upon the viewer's perspective. An operating system designer may view VMH as an interprocess communication facility. A communications engineer may regard it as a message-switching utility. Programmers often regard it as a terminal input/output (I/O) package, while users at the terminals may see it as a somewhat specialized command system. All these functions are indeed performed for the Video Graphics System by the software package called VMH, the Video Graphics System Message Handler. VMH provides multiplexed communication control in such a way that human users and computer programs can selectively determine the destinations of messages they send and the sources of messages they receive.

VMH is a part of VOS, the Video Operating System, which in turn is a part of VGS, the Video Graphics System.(1,2) VGS is a multi-access, terminal-oriented computing system, based on closed-circuit TV technology with analog/digital scan conversion, that permits each user to communicate upon request with any of several computers and any of a number of services available in these service machines; each terminal provides a choice of input devices, graphical display and input, and a full range of interaction from remote job entry to dynamic graphics.

From a hardware viewpoint, VGS consists largely of image generation and storage devices, controlled by an IBM 1800 process control computer, that sends the stored images over high-capacity coaxial cables to television monitors at suitable refreshing rates. Each monitor is the principal output device in a specially designed, inexpensive user terminal that contains an extended-character-set keyboard and optional, pluggable input or output devices, including the Rand Tablet.(3) The control computer receives inputs from terminals and routes them to one of the service computers. It also receives outputs from the service machines directed to the terminals--usually, order codes for the image generator. The hardware configuration is shown schematically in Fig. 1.

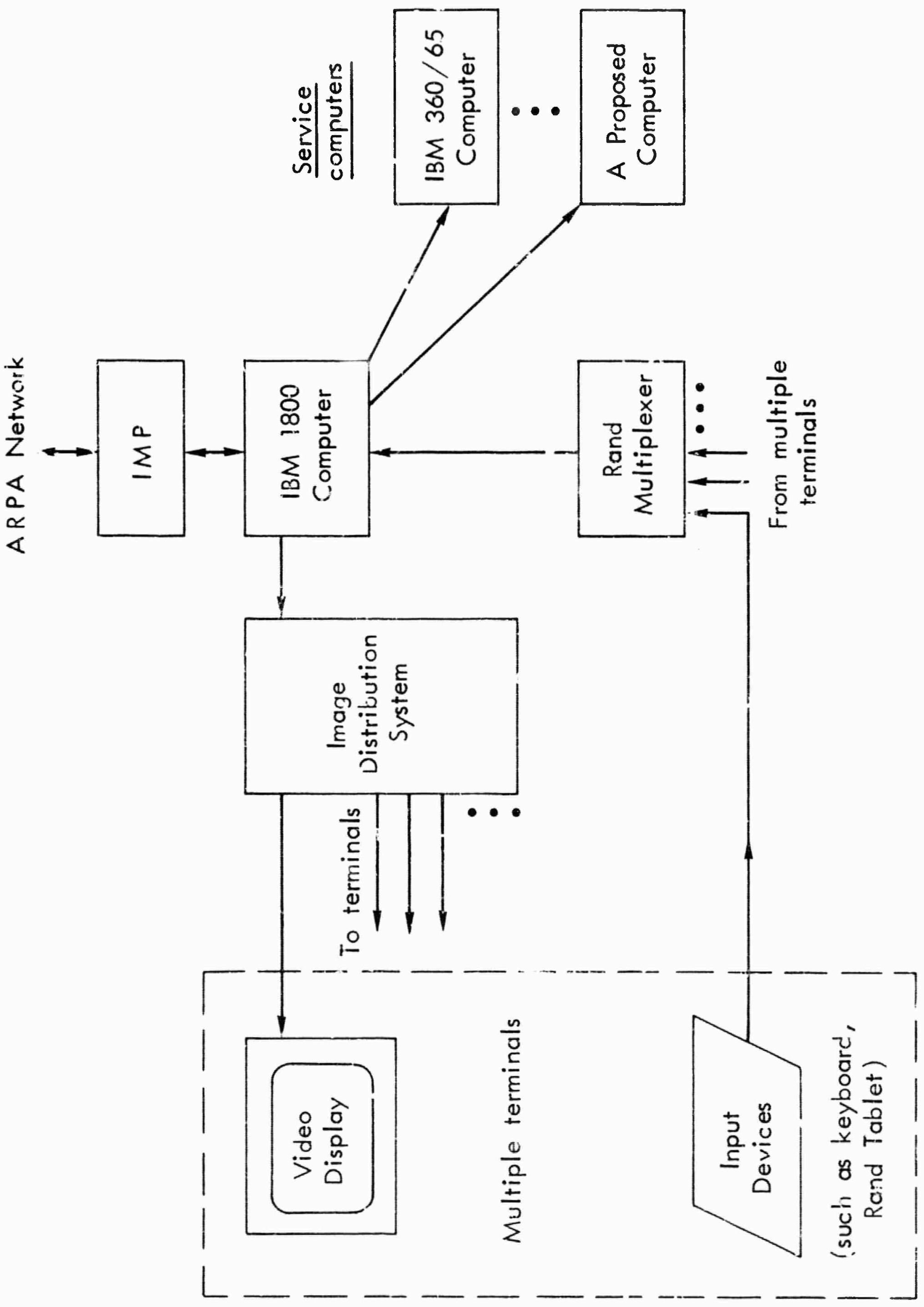


Fig. 1 —Rand Video Graphics System hardware configuration

VGS includes two software packages: the 1800 software and VOS, the Video Operating System, which consists of VMH plus a very small amount of command processing and the time-slicer. VOS is an extension of OS/360 within each connected IBM 360 service machine.

At Rand, the 32-terminal VGS system interfaces with the ARPA Computing Network(5) so that users have access to the entire range of Network facilities. VMH acts as the communications controller, handling all messages among terminals, programs in the service machines, and the Network: program/program, program/terminal, terminal/program, Network/program, and program/Network. Multi-access subsystems and the Network's Interface Message Processors are each treated as single users for the purpose. (Terminal/terminal communication should be used only in connection with an application program.)

As now implemented, VMH consists of a set of interrelated routines, some of which are supervisor calls (SVC) and some of which are link edited together and run as asynchronous event-driven subtasks of an OS/360 job step. VMH facilities are available to programs via a set of Assembler Language macro instructions, the expansions of which include SVC instructions. The VMH software resides in an OS multiprogramming environment operating under IBM MFT II or MVT. When initiated as an OS task, it has the highest dispatching priority of all OS tasks except for System tasks such as the Reader-Interpreter, Write, and Master Scheduler. Some components of the VMH code reside in the System Nucleus or are brought into core as needed.

The present Report is not a user's guide. As a self-contained presentation, it will be of interest to programmers and users who are already familiar with the system. Those who are fully familiar with the internal documentation of VMH, however, should read with care; many of the procedure and data names have been changed in this Report for the sake of better indicating their purpose. (These names are strings of capital letters that may include a combining underscore, e.g., INTRINSIC_ID.) Also, because of the logical complexity of VMH programming, some minor implementation features have been ignored in this description and some have been much simplified.

This presentation is intended primarily for those who might influence the design of systems of similar purpose--systems for providing communications between or among currently executing computer programs and/or remote terminals, especially in an OS/360 operating environment

II. HISTORY AND CONTEXT

During the mid-1960s, when the potential usefulness of interactive computer graphics was becoming widely appreciated, but general-purpose graphic displays were very expensive, The Rand Corporation undertook to find a method of making them economically accessible on a multiple-user basis. The result was the Video Graphics System, described above. Such a system would obviously require two kinds of supporting software: (1) graphic support programs for use in processing pictures and user control information, and (2) programs to provide I/O services between application programs and VGS terminals.

The graphical support programs could be, and were, adapted from previous Rand software, written for use with the few commercially available graphic display devices.(3,4) The I/O service programs had to be designed from the beginning. The name given this software was Video Message Handler, abbreviated as VMH, and the acronym was retained even after the purpose of the software had broadened from pure graphical input/output to general communication among programs and terminals.

While the first need was for software that would permit the Rand graphical routines to be accessed from a VGS terminal, it was realized that an ample supply of inexpensive terminals would eventually create a demand for interactive services, such as on-line programming and text editing. Because of the nature of the service machines (IBM 360s) and their operating system, the Rand system designers thought that these interactive services should be developed as separate subsystems, each accessible to many users at a time, and each running under control of the primary operating system. We planned, therefore, to provide a means of selecting from these services and switching smoothly between them from any terminal, and this required the facility for a single terminal to communicate concurrently with more than one program. We realized at the time that this was a difficult goal to achieve, even in a well-designed, integrated operating system.

Certain desired applications, such as on-line debugging, required the ability to communicate across the boundaries of OS/360 job steps--

an action OS/360 is expressly designed to prevent. (A job step is the primary unit of OS/360 processing.) This form of communication became part of the VMH concept.

VMH was broadened also to provide computer support for interactive group problem-solving or gaming. Rand staff members had long felt that such applications should be able to utilize groups of interconnecting programs, as desired, rather than be limited to multiple-access use of single programs.

With program-to-program and terminal-to-program communication incorporated in the design of VMH, we considered providing direct terminal-to-terminal communication, independent of any application program, perhaps using a disk storage "mailbox." This idea was rejected because such a facility would require a fairly sophisticated extension, which could better be implemented as a separate program utilizing VMH.

Thus, the VMH design became that of a computer message exchange serving two categories of users: VGS terminals on one hand, and programs running under control of OS/360 on the other. The two categories of users are treated almost exactly alike with respect to communication control, but quite differently with respect to actual message handling.

THE OS/360 ENVIRONMENT

The final shape of VMH--although not its conceptualization--was very strongly influenced by the fact that it was implemented on a System/360 computer. The version described here runs under control of the MVT (Multiprogramming with a Variable Number of Tasks) version of OS/360.(6)

For those unfamiliar with OS/360, its salient features will be briefly described. OS/360 processes concurrently up to 15 separate "job steps"; the number is limited by System/360 architecture. Each job step is treated as an independent process or "task," and can store information only in its own specified, fixed-size region of primary (core) storage. When a job step is initiated, the system assigns it a unique four-bit "protection key"--a number between 1 and 15--and the "storage key" of every 2048-byte (2 K) block in its region of core is set to that value. Only when the protection key of a task matches the storage key of a block may that task store into that block of storage. (The only exception is when the protection key has a value of zero--a value that OS/360 reserves for itself.)

Each storage block has a fifth bit which, when set to 1 by a program instruction, prevents any task from fetching as well as storing information there without the appropriate protection key. There is no facility for one job step to communicate with another except via the file system.

The processor contains a register called the Program Status Word (PSW), which contains the protection key of the task currently executing. As with most operating systems, certain machine instructions can be executed only when the processor (i.e., the current task) is in the "supervisor" rather than the "problem" state, as indicated by a bit in the PSW. Among these are instructions to change the value of storage keys, enable or disable machine interrupts, or otherwise change the PSW contents.

Under the MVT version of OS/360, a task may request the creation of one or more subtasks of itself. A job-step task and its subtasks communicate and synchronize with each other and with OS/360 services via flagwords called Event Control Blocks (ECBs).

A task may issue a WAIT request to OS/360, specifying the address of an ECB or a list of ECBs; this task is then put into the WAIT state until another task issues a POST request recording the occurrence of the awaited event, and specifying one of these ECBs. (Actually, a task may request a WAIT for the POSTing of any one of a list of ECBs.) The POSTing task may also specify up to 30 bits of information to be placed in the ECB. In the case of certain operating system services, including the execution of channel I/O programs, OS/360 does the POSTing.

Each task has a dispatching priority, represented by an integer.

OS/360 switches the processor from one executing task to another on two occasions:

1. When the executing task issues a WAIT for which no satisfying POST has yet been issued, OS/360 selects the ready task of highest priority, if any.

2. When an event is detected (via a POST issued by the executing task or via some type of interrupt), which results in a POSTing that satisfies the WAIT of a task of higher priority than the executing task, the task with the satisfied WAIT is given control.

For each task there may be a queue of one or more nonreentrant procedures, called "attention routines" (ARs), to be scheduled for execution upon the occurrence of certain previously specified events. When an event occurs for which an AR is specified, a data element representing the event, an "interrupt queue element" (IQE), is placed in a list for that AR. Whenever a task is dispatched, control is given to the first AR in the queue for which an IQE exists or which has been previously interrupted. If there are no such ARs, the task is resumed normally.

THE RESULTING FORM

The basic decisions arising from the considerations above are reviewed here.

1. VMH runs basically as a user job step, in order to utilize OS/360 services most conveniently and to maintain independence of the particular version of OS.

2. VMH has a dispatching priority higher than any task that could otherwise interrupt it, in order to meet the severe real-time demands of interactive graphics and to guarantee the integrity of the VMH data base.

3. Because VMH must be able to pass messages across storage region boundaries, it must execute with a protection key of zero at least part of the time.

4. Data associated with a program using VMH are stored in that program's region; but, in order to avoid accidental alterations to the information VMH needs for control purposes, that information must be store-protected from its own program as well as from all others and accessible only to VMH. Therefore, part of each program's region has a storage key of zero.

5. Because VMH services are accessible to programs via subroutine calls and because these subroutines must access the VMH data base, they must execute with a protection key of zero. They must be non-interruptible when accessing it, to prevent conflicts with asynchronous tasks of VMH updating the data base. Therefore, these subroutines were implemented as supervisor calls (SVCs).

6. However, during initial performance evaluation it was discovered that more efficient use of OS/360 services is obtained by branching to them directly, rather than calling the supervisor. This fact, plus the need to request such services on behalf of a using program, required the ability to disable and reenable hardware interrupts. To achieve these goals, it was necessary for VMH job steps to execute in the supervisor rather than the problem state. Thus VMH can switch into a protection key of zero only when necessary, avoiding some of the harder-to-trace addressing errors and protecting the Operating System itself. This is a great advantage in terms of checkout and reliability.

III. MESSAGE SWITCHING

At the highest level, VMH is a symmetric exchange, uniformly serving a set of users wishing to communicate selectively with each other. Figure 2 shows the basic concept. A "VMH user" is defined as either a VGS terminal that has logged on to VMH or an OS/360 user task--referred to here as a program or user program--that has connected itself to VMH. The user program may be either an OS/360 job step task or one of its descendant subtasks. We consider the terminals, rather than the persons at the terminals, as the VMH users, because only the terminals can be immediately and positively identified by the system.

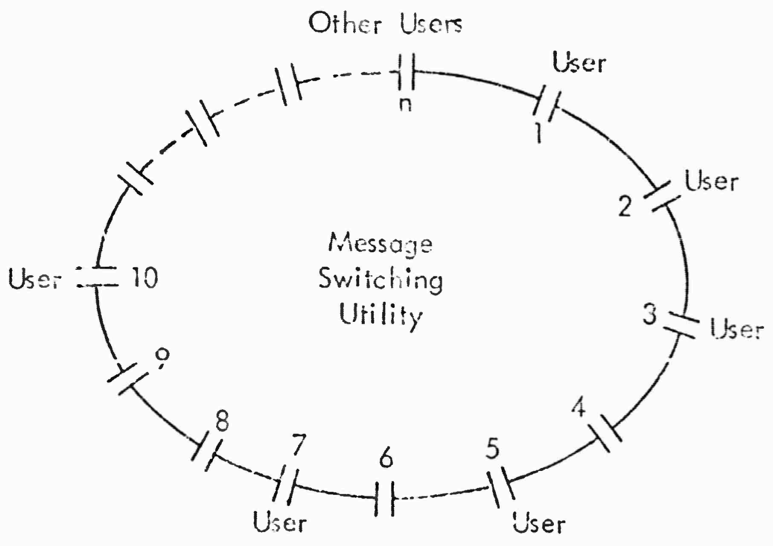


Fig.2 — The symmetric message-switching utility

We may imagine VMH as an agent working inside a closed box containing a number of ports, one for each potential user. The agent has limited knowledge of the users. Users may know of and deuc e each other by *a priori* external names. The agent's guidelines are that:

1. A message presented by a user is to be delivered to and accessible by specified other users, and only by its intended receivers.

2. A message is to be delivered to a user only if it is from one of the recipient's desired sources.

The first requirement offers protection against unauthorized access to information, a property common to "secure" systems. The second offers protection against unwanted information, a form of privacy frequently overlooked. So far, the limitation on unwanted access has not been much used; most current programs are open to all messages directed to them. However, its existence enables VMH to serve possible future communication systems in which this characteristic might be vital.

To carry out these functions obviously requires that users must be identified in such a way as to be clearly distinguished, and an appropriate communications policy must be enforced.

USER IDENTIFICATION

At the time each terminal or program joins the system, it must give the system information called its INTRINSIC_ID, by which it will be identified. Then or later, the user may also supply a NAME by which other users can refer to it. (Naming is necessary only when communicating with terminals or with programs that are selective in their communications; it is unnecessary for communicating only with multi-access subsystems or programs that accept all messages from terminals.) VMH assigns each user an integer USER_NUMBER, which serves as an index to all information associated with that user, as described in Section V. The core of this information is the USER_DESCRIPTION.

When a program joins VMH, an SVC procedure called CONNECT establishes the program as a VMH user by creating and initializing a USER_DESCRIPTION in the protected area of the program's region (called the DESCRIPTION_AREA.) The INTRINSIC_ID is taken as the address of the "Task Control Block" (TCB), a description created and maintained by OS/360 for each existing task in a fixed location in its own region of storage.

The TCB address of the currently executing task is easily obtained (via a four-instruction sequence) by the CONNECT procedure and by every other VMH service SVC, making initial and subsequent identification a simple matter. The assigned USER_NUMBER of a connecting program is stored in an unused field of its TCB so that it can be obtained directly by the service procedures. Section VII describes implementation of the joining and identification procedures.

THE MESSAGE PATH SCHEME

The model for communication control implemented in VMH is derived from the axiom that any two users should have equal control over any communication link between them.

Associated with any two users, A and B, are four unidirectional "links," represented collectively by either of the symbols

$$A\overset{wx}{\underset{zy}{}}B \qquad \text{or} \qquad B\overset{zy}{\underset{xw}{}}A$$

where w represents a link originating with A and directed toward B, x a link directed toward B and terminating with B, y a link originating with B and directed toward A, and z a link directed toward A and terminating with A (Fig. 3). The pair wx is a "message path" from A to B, and the pair yz is a message path from B to A. Links w and z are A's "link pair" with B, and x and y are B's "link pair" with A.

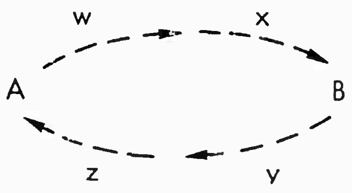


Fig. 3--Message paths between A and B

Each link may be null, represented symbolically by "o"; or it may exist, in which case it is symbolized by "-". Thus, a message path can have one of four states, called its PATH_STATUS.

The status of the paths between users A and B is initially "oo" or "null." The VMH communication control policy is that messages can go from A to B (or from B to A) only if the status is "--" (the path exists). Communication between two users can be attempted, established, rejected, or broken by any one of the following rules:

1. Either user can always request either link of his link pair with the other user to be "created."

2. Either user can always request that either message path be deleted, i.e., reset to null.

3. Each user is always notified of any change in PATH_STATUS effected by the other.

The dialog protocol for creating a message path is straightforward. By means of the above rules, users can signal their desires and intentions regarding communication with each other.

In practice, both message paths between a pair of users are usually created or deleted together. To facilitate this, "dual requests" are permitted if the two paths have "dual status," which means that the links of each user's link pair have the same status.

It proved necessary to introduce an auxiliary state for the terminating link of a message path, by which the destination user could temporarily block the path. This temporary blocking state is symbolized by "+". The source user is informed of this temporary status. Thus, there are five possible values for message path status, as listed in Table 1. The alternative symbols "NL", "ID", etc., are verbally (though not visually) more convenient and will be used in subsequent text.

Table 1

MESSAGE PATH STATUSES

Symbol		Meaning
oo	NL	Null
o-	ID	Initiated by the destination
-o	IS	Initiated by the source
-+	CB	Complete but (still) blocked by the destination
--	CU	Complete and unblocked

Besides serving the requirements of communication control, the message path scheme allows messages to be multiplexed and routed on the basis of implicit rather than explicit information; lacking specific addresses, a message is routed to all users that have a CU message path from the sender.

PATH_STATUS change requests and their effects are listed in Table 2. PATH_STATUS is controlled by, and changes are requested of, a VMH service known as the MESSAGE_PATH procedure. For n users, the representations of PATH_STATUS form an n x n matrix, with the element in row i, column j signifying the status of the path from user i to user j. The matrix is stored by rows. Each row, called a PATH_STATUS_TABLE, is stored as part of the USER_DESCRIPTION for that user.

Table 2

MESSAGE PATH REQUESTS BY USER A

Requests	Path Status	
	Before	After
Single Requests		
Initiate the path to B	$A^{oo}B$	$A^{-o}B$
	$A^{o-}B$	$A^{-+}B$
Initiate the path from B	$A_{oo}B$	$A_{-o}B$
	$A_{o-}B$	$A_{+-}B$
Unblock the path from B	$A_{+-}B$	$A_{--}B$
Block the path from B	$A_{--}B$	$A_{+-}B$
Delete the path to B	$A^{wx}B$	$A^{oo}B$
Delete the path from B	$A_{zy}B$	$A_{oo}B$
Dual Requests		
Initiate the paths to and from B	$A^{oo}_{oo}B$	$A^{-o}_{-o}B$
	$A^{o-}_{o-}B$	$A^{-+}_{+-}B$
Delete the paths to and from B	Any dual status	$A^{oo}_{oo}B$

The diagonal elements of the matrix are meaningful. Since a terminal and its TERMINAL_AGENT control facility (described in Section VI) are one user to VMH, the ability to "send a message to himself" enables the TERMINAL_AGENT to communicate with the man at the console.

The notification required by the rule of delivering messages only to willing recipients is carred out by means of the Message Path Attention Routine (MPAR), one of which is associated with each user. Whenever a user requests, and thereby effects, a change in PATH_STATUS, the MESSAGE_PATH service sends a request to OS/360 scheduling the MPAR of the other user for execution. Most communication control activities are carried on by the MPARs, independently of the message-handling operations. In VMH as a whole, message switching and message handling are independent functions; their only interface is the PATH_STATUS matrix, as is seen in Fig. 4.

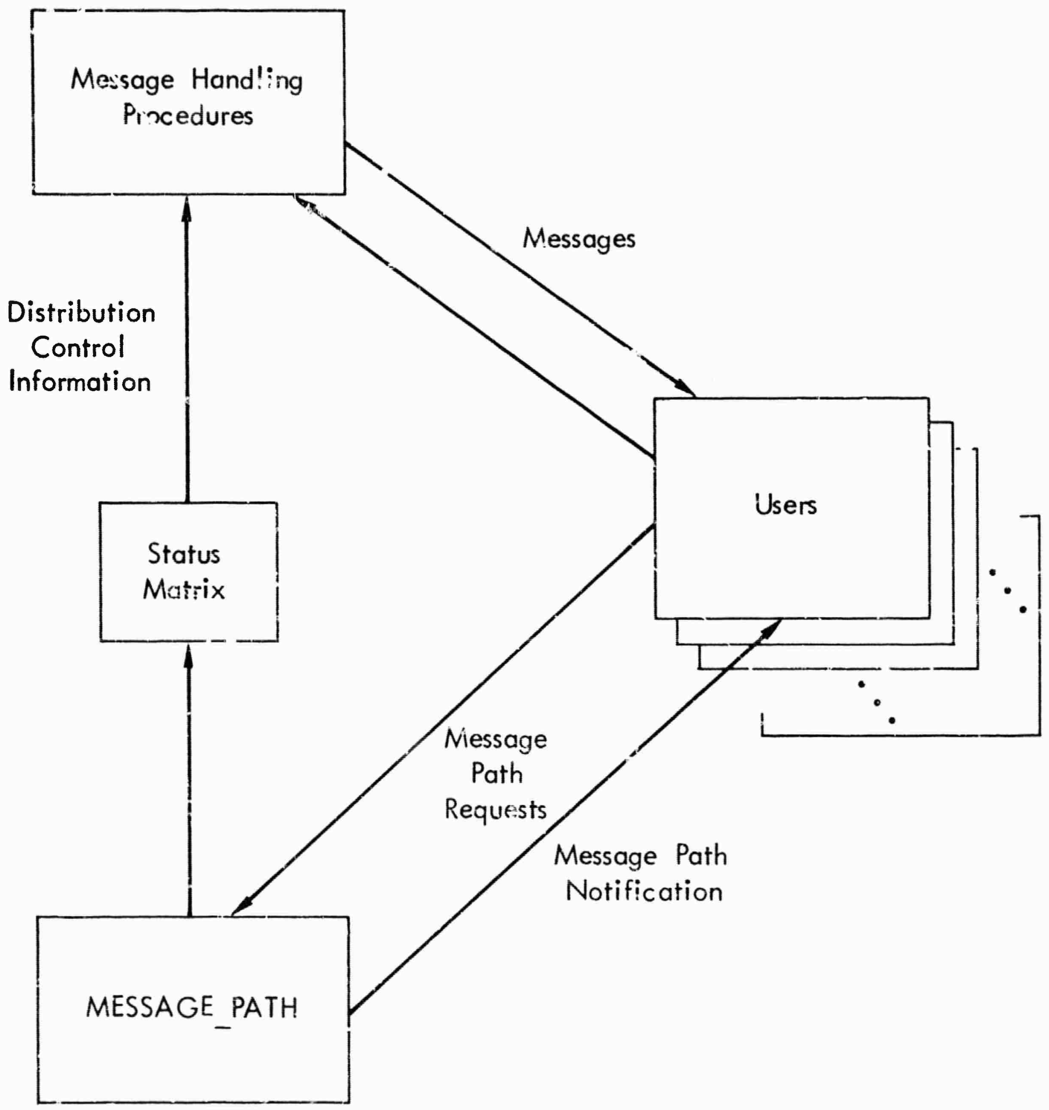

Message Handling Procedures
Messages
Distribution Control Information
Users
Status Matrix
Message Path Requests
Message Path Notification
MESSAGE_PATH

Fig. 4—Interface between message handling and message path control

IV. MESSAGE HANDLING

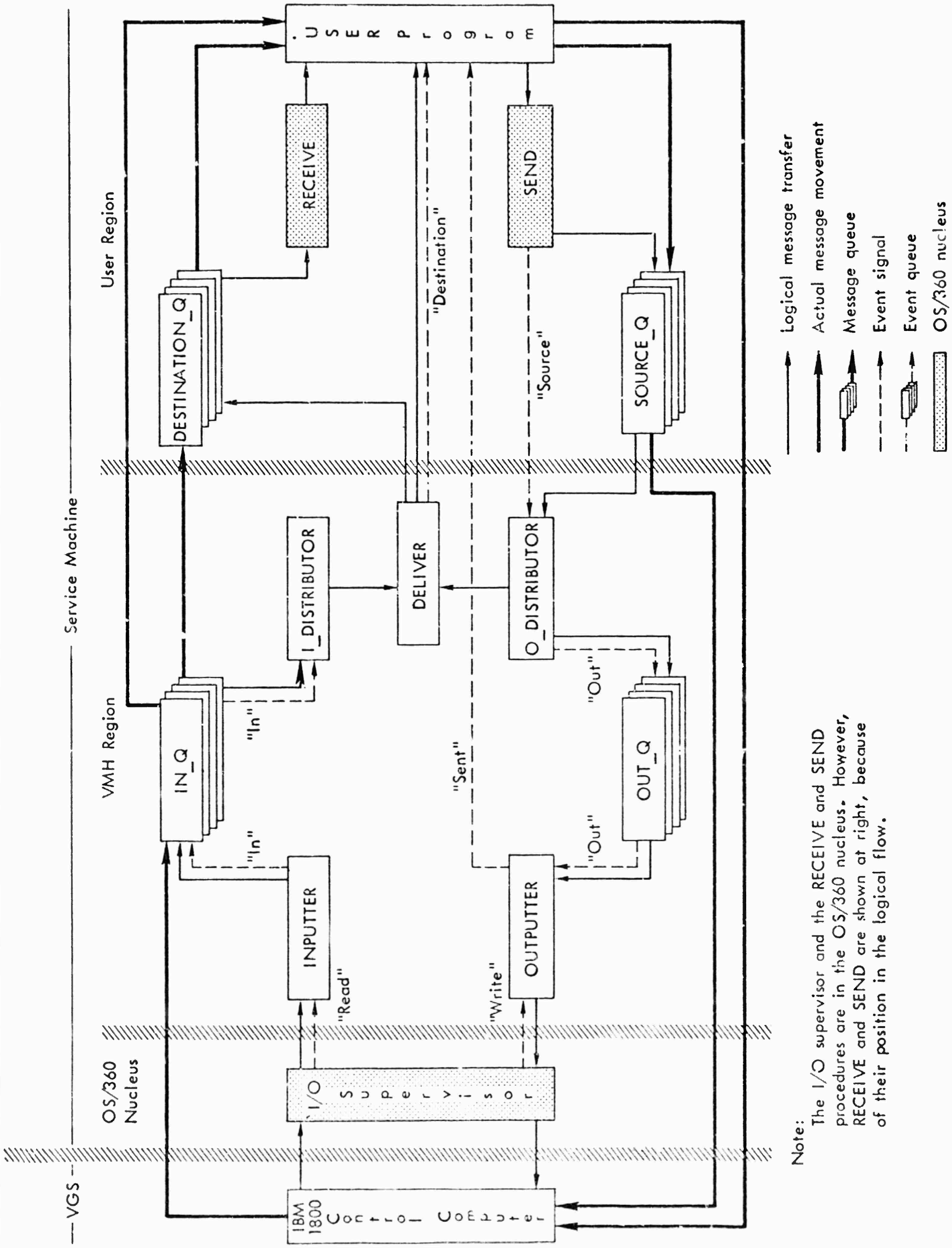
VMH message handling activity is triggered either by the receipt of a message from a VGS terminal via the VGS control computer or by the sending of a message by a program. From the VMH viewpoint, these are asynchronous events, occurring in no fixed sequence, and they are handled by means of first-in-first-out queues (described in more detail below). The most important consideration in VMH internal design is message throughput, achieved by minimizing both the processing time per message and the delays of messages entering or leaving the system.

LOGICAL INTERRUPTS

Since messages may arrive faster than they can be processed, message arrival must be allowed to interrupt other processing. However, complete interruptibility is generally expensive because of the necessity of saving and subsequently restoring the contexts of the interrupted processes. The technique adopted in VMH, therefore, was to design processes so as to provide logical interrupt points--where context is minimal and control can be switched rapidly--and to allow one routine to interrupt another only at these points. The result is a set of non-interruptible but short asynchronous procedures, communicating with each other and driven by a corresponding set of "event queues," described below.

VMH itself can be interrupted by events that cause OS/360 to switch control to higher-priority tasks, but these must be System tasks and are presumably of short duration. VMH priority is such that it is never interrupted by user tasks.

The asynchronous message-handling procedures, the types of event that drive them, and the events they signal are given in Table 3. The RECEIVE and SEND procedures are driven by calls from user programs and are implemented as service SVCs. They are described in Section V. The other four procedures are implemented as subtasks of the VMH job step and are described in Input, Output, and Routing, below. Fig. 5 shows the overall structure.



Note:
The I/O supervisor and the RECEIVE and SEND procedures are in the OS/360 nucleus. However, RECEIVE and SEND are shown at right, because of their position in the logical flow.

Fig. 5—Overview of message handling, showing flow of messages and flow of control

EVENT QUEUES, MESSAGE QUEUES, AND SEQUENCING

"Asynchronous" means that there is no fixed sequence in which the above-mentioned procedures receive control. Order-independence is effected by a set of event and message buffers or queues (Qs), shown in Fig. 5. With the optional exception of the DESTINATION_Q (see below), they are all first-in-first-out queues.

The IN_Q is a queue of event descriptors in the form of pointers to messages that have been satisfactorily read in from the VGS system and are now in the input buffer. Thus, the term IN_Q refers to an event queue as well as to a message queue, both in the VMH region.

For *each* user program there is a DESTINATION_Q, which consists of messages delivered to that program but not yet accessed and released by it. The DESTINATION_Q resides in a special part of the program's region called the MESSAGE_AREA. Unlike the DESCRIPTION_AREA, the MESSAGE_AREA is not read- or write-protected from the program. Messages from terminals are placed in the DESTINATION_Q by the I_DISTRIBUTOR, and messages from other programs are placed there by the O_DISTRIBUTOR, using a common procedure called DELIVER (described below).

Access to messages in the DESTINATION_Q is specified by the user program via calls on the RECEIVE SVC. It is a message queue only and not an event queue because the program has options regarding if and how it will be synchronized with and notified of message arrivals.

For each program there is also a SOURCE_Q consisting of descriptions of messages whose sending has been requested via calls on the SEND SVC. The SOURCE_Q resides in the program's DESCRIPTION_AREA. It is not an event queue because when the SOURCE event is POSTed by the SEND procedure, the VMH job step is immediately dispatched by OS/360 (because of VMH's higher priority) and the message is processed by the O_DISTRIBUTOR at that time. The SOURCE_Q is necessary because of the possibility of a backlog of messages awaiting transmission to terminals.

For each VGS terminal destination of a message enqueued in the SOURCE_Q, there is an entry in an event queue called the OUT_Q, consisting of the USER_NUMBER of that terminal, a pointer to the SOURCE_Q entry, and a pointer to an ECB optionally specified by the sending program. Each OUT_Q entry is stored in the sending program's DESCRIPTION_AREA, but a separate queue of pointers to current OUT_Q entries from all programs resides in the VMH region and is considered the base part of the OUT_Q.

The sequencing logic among the four VMH subtasks and the two message-handling services is completely defined by the occurrence of events of the types listed in Table 3 and the relative priorities of the various procedures. The subtasks are ordered from highest to lowest priority:

INPUTTER
OUTPUTTER
I_DISTRIBUTOR
O_DISTRIBUTOR

(A fifth subtask, of lower priority, is the LOGON_PROCESSOR, described in Section V, the Terminal Interface). Because they execute as part of user tasks, the RECEIVE and SEND procedures have lower priority than any VMH subtask.

OS/360 does not provide a convenient means for the enqueuing of event notices. If an ECB is associated with an event, and if an occurrence of that event is POSTed before the ECB contents resulting from the POSTing of the previous occurrence have been read, the data of this subsequent POST will be lost. For this reason, the VMH subtasks are controlled by a very simple, specially built task supervisor (the CONTROLLER), which does provide the needed event queuing capability. The CONTROLLER is described in Section VII, under Subtask Control.

A subtask is dispatchable whenever there is an event notice in the event queue by which it is driven. The READ, SOURCE, and WRITE events, which drive the INPUTTER, Q_DISTRIBUTOR, and phase two of the OUTPUTTER, respectively, may be thought of as having queues of capacity 1.

Table 3

MESSAGE HANDLING EVENTS

Procedure	Driving Event		Signalled Event
	Name	Description	
INPUTTER	READ	Completion of a channel input operation from VGS.	IN
I_DISTRIBUTOR	IN	An input message successfully read.	DESTINATION
O_DISTRIBUTOR	SOURCE	A message is ready for distribution to other users.	DESTINATION (for program destinations) OUT (for terminal destinations)
OUTPUTTER (Phase 1)	OUT	A message is ready for transmission to a terminal.	(A channel output request to OS/360)
(Phase 2)	WRITE	Completion of a channel output operation to VGS.	SENT
RECEIVE	(SVC)	Request by a program to receive a message	(None--request is filled or saved)
SEND	(SVC)	Request by a program to send a message	SOURCE

The VMH job step is dispatched by OS/360 whenever any VMH subtask is dispatchable. VMH is entered through the CONTROLLER, which dispatches the highest priority dispatchable subtask. When a subtask returns, the CONTROLLER issues a WAIT to OS/360 if there are no more dispatchable subtasks.

INPUT, OUTPUT, AND ROUTING

We now briefly summarize the handling of messages as they are received from the VGS system, sent by programs, and transmitted to the VGS system. At the outset, the INPUTTER initiates a READ request on the input channel from VGS. When VGS has a message from a terminal to be sent to this service machine, it initiates a WRITE on that channel and transmission takes place.

When the operation is completed, the READ event is POSTed (by the I/O supervisor of OS/360) and the INPUTTER receives control (within an interval of time no longer than the maximum time needed to execute any VMH procedure--see Table 4). If the message has been read successfully, the INPUTTER signals an IN event and initiates another READ.

When the I_DISTRIBUTOR is dispatched, it delivers a copy of the message to each program for which the path from the source terminal, as determined by the terminal's PATH_STATUS_TABLE, has the status CU. For each such program, the DELIVER subroutine (described below) is called. When the I_DISTRIBUTOR has routed the message to all its destinations, the message is deleted from the IN_Q.

The SEND SVC is described in Sending, Section V. Primarily, it constructs and enqueues a SOURCE_Q element.

The O_DISTRIBUTOR performs the routing of each message sent by a program. The message may be addressed either explicitly, via a DESTINATION_LIST provided by the sending program, or implicitly, via its PATH_STATUS_TABLE. In the former case, each given destination must be one for which the message path from the program has the status CU; in the latter case, the message is routed to each user to which there is a CU path. Routing is handled differently, depending on whether a given destination is a program or a terminal. If it is a program, the DELIVER subroutine is called (see below). If it is a terminal, an OUT_Q element is constructed and an OUT event signalled.

Phase 1 of the OUTPUTTER, which is dispatched for each occurrence of an OUT event, initiates a write operation on the output channel to VGS. On the basis of information in the SOURCE_Q element, OUTPUTTER then returns, waiting for VGS to respond with a read on that channel and for completion of the resulting transmission. When the operation is completed, a WRITE event is POSTed (by OS/360), making the OUTPUTTER again dispatchable.

In phase 2, the OUTPUTTER checks to make sure the message has been transmitted successfully and, if so, deletes the OUT_Q element. If this was the last OUT_Q element associated with a SOURCE_Q element, the latter is also deleted. Before returning, the OUTPUTTER POSTs a SENT event for the sending program, if so requested.

DELIVERY

The DELIVER subroutine is called by either the I_DISTRIBUTOR or the O_DISTRIBUTOR to deliver a message to a given program destination. It can be described only in terms of possible prior activity of the RECEIVE and SIMAR procedures, presented in Section V, under Receiving.

DELIVER first examines the program's RECEIVE_TABLE, if any, to see if the message has been requested. If so, the access mode is examined. If the AREA parameter is given, the message is copied directly into the specified area of the program, bypassing the MESSAGE_AREA. If AREA is omitted, the message is copied into the MESSAGE_AREA and enqueued on the DESTINATION_Q; any message having the same DESIGNATION, which has already been accessed by the program but which has not been RETAINed, is deleted from the DESTINATION_Q. The RECEIVE_TABLE entry is deleted and the ECB, if given, is POSTed.

If the message has not been requested in a RECEIVE call, it is copied into the MESSAGE_AREA and enqueued on the DESTINATION_Q. The program's IMAR_TABLE (Incoming Message Attention Routines) is examined. If an IMAR has been specified for messages having the given DESIGNATION, it is scheduled (via a request to OS/360).

V. THE USER PROGRAM INTERFACE

From the point of view of a user program, VMH consists of a set of services provided in the form of calls on SVC routines. The calls and their parameters are designed for flexible message reception and transmission, with straightforward use for simple applications. Thus, most of the calls contain optional parameters with default values.

The program services are reviewed below, together with the principal activities of each corresponding VMH procedure. Optional parameters are enclosed in brackets.

CONTROL SERVICES

The procedures for joining VMH, obtaining another user's USER_NUMBER, changing message path status, and disconnecting from VMH are described here.

Connecting to the System

The call that establishes a task as a VMH user--a prerequisite to the use of any other services--is:

```
CONNECT(NAME,MPAR,[OVFLAR],[REPLACE],[D_AREA_LIMITS],
[M_AREA_LIMITS],[AREA_OWNER],RESULT),
```

where

NAME is a unique character string by which this VMH user can be referenced by others.

MPAR is the entry point of a Message Path Attention Routine.

OVFLAR is the entry point of an optional attention routine to be scheduled by VMH whenever a new "overflow condition" is detected. (See RECEIVE call, below).

REPLACE specifies how incoming messages are to be handled when an overflow condition exists. If it is omitted, such messages are discarded. If given, such messages replace the oldest unaccessed message or messages in the DESTINATION_Q, which are thus lost.

D_AREA_LIMITS and M_AREA_LIMITS specify the minimum and maximum sizes of the DESCRIPTION_AREA and the MESSAGE_AREA, respectively. This gives the program complete control over the portion of its region to be allocated to these uses. If these limits are omitted, standard values are used.

AREA_OWNER, the name of another VMH user, a task of the same job step as the calling task. If this parameter is given, the previous two are ignored and the DESCRIPTION_AREA and MESSAGE_AREA of the named user are shared.

The RESULT parameter, a part of most calls, indicates to the user whether the call was executed normally and/or successfully.

"Successful" and "normal" are not synonymous because, in certain cases, part of the request may be fulfilled even though the entire request cannot be completely satisfied. In the case of CONNECT, one of the following abnormalities may be indicated:

1. Not enough space is available in the program's region for an initial DESCRIPTION_AREA and MESSAGE_AREA;

2. The given NAME is not unique;

3. There is no user in the same job step having the name given as AREA_OWNER.

The principal operations of the CONNECT procedure for a CONNECTing program are the following:

1. Assigning a USER_NUMBER.

2. Checking the uniqueness of the NAME.

3. Finding the AREA_OWNER, if any.

4. If shared areas are not to be used, obtaining space for the DESCRIPTION_AREA and MESSAGE_AREA, and initializing these areas for subsequent suballocation.

5. Constructing and initializing a USER_DESCRIPTION.

6. Declaring the existence of the specified attention routines to OS/360.

7. POSTing the NEW_USER event for any users currently waiting for the appearance of a user with this NAME.

Obtaining the Number of Another User

For the sake of efficiency, every reference to another user must occur in the form of its USER_NUMBER, except in CONNECT and in this call, which obtains the USER_NUMBER of another user and returns it as RESULT:

GET_USER_NUMBER(NAME,RESULT,[ECB]).

A zero in RESULT indicates that no such user exists (yet). The ECB, if specified, will be POSTed by VMH with the USER_NUMBER, if and when such a user CONNECTS. Thus, a program may synchronize on the appearance of another program or terminal as a VMH user.

Changing Message Path Status

To change the status of any message path to or from a program, the call is:

MESSAGE_PATH(USER_NUMBER,REQUEST,RESULT).

The possible requests are listed in Table 2. The RESULT can indicate that USER_NUMBER refers to a non-existent user, or that the request was inappropriate for a path having the current status. The functional logic of the MESSAGE_PATH procedure includes:

1. Checking the validity of the given USER_NUMBER.

2. Checking the appropriateness of the request.

3. If a path to the caller is being deleted, updating its RECEIVE_TABLE and IMAR_TABLE as necessary. (See Receiving, below).

4. Changing the proper PATH_STATUS_TABLE entry.

5. If the request is a dual request, repeating steps 2 through 4 for the corresponding path.

6. Constructing an MPAR parameter area and scheduling the other user's MPAR.

Disconnecting

The call

DISCONNECT

results in deletion of the calling program as a VMH user. The DISCONNECT procedure (1) terminates any other VMH users sharing the DESCRIPTION_AREA and MESSAGE_AREA owned by this user; (2) calls MESSAGE_PATH on behalf of the program for all non-null paths to or from the program; (3) deletes the program's attention routine declarations; and (4) returns the space for the DESCRIPTION_AREA and MESSAGE_AREA if these are owned by this VMH user.

RECEIVING

The call that accesses a particular message in the program's DESTINATION_Q is:

RECEIVE([DESIGNATION_LIST],[ORDER],[AREA],[ECB],RESULT).

The message accessed is the first message found in the DESTINATION_Q at the time of the call--or, if none is found, the first message which later arrives--the type and source of which match those in some entry of the DESTINATION_LIST. The RECEIVE request is then "satisfied"; until then it is "pending," and is represented by an entry in a RECEIVE_TABLE, stored in the program's DESCRIPTION_AREA. The RECEIVE_TABLE entry consists of the saved DESIGNATION_LIST and AREA and ECB parameters of the call.

Each message has a TYPE identifier. For messages from terminals, TYPE must be one of a standard set of VGS types indicating the function of the message. Some of the more common examples follow:

Standard VGS Message Types from Terminals

Keyboard character
Keyboard line(s)
Rand Tablet data
Program attention

Standard VGS Message Types Sent to Terminals

Display line(s) and unlock keyboard
Enter single-character keyboard mode
Display picture
Unlock Rand Tablet
Determine stylus position
Lock Rand Tablet
Audio message (not yet implemented)

DESIGNATION_LIST lists entries of the form (USER_NUMBER,TYPE). An entry may contain a zero for either component, which is interpreted as "any source" or "any type," respectively. If the DESIGNATION_LIST parameter is omitted, the message accessed is the first one found, or, if none is found, the next arriving message. In other words, a single designation, (o,o), is assumed.

ORDER specifies that the DESTINATION_Q is to be searched beginning with either the first (oldest) or the last (most recent) message. The former option is the default. Thus, the DESTINATION_Q can be used as either a first-in-first-out or a last-in-first-out queue.

AREA is the address of an area into which the message is to be moved when the RECEIVE request is satisfied. For the protection of the program, the area must contain, as a header, an indication of its own length. If the parameter is omitted, the message remains in the DESTINATION_Q when accessed; the message is deleted when the next message having the same designation arrives. It may be retained indefinitely, however, by use of the RETAIN call (see below).

ECB is the address of the ECB to be POSTed when the RECEIVE request is satisfied--whether or not it is satisfied immediately. (This is the DESTINATION event mentioned in Section III.)

A subsequent call from the same program causes an entry in the RECEIVE_TABLE to be replaced if either the AREA or the ECB parameter is the same; otherwise, a new entry is added.

On return from the RECEIVE call, the RESULT parameter contains a pointer to the message (if one was found) or a zero indicating that the request is pending. If neither, one of the following abnormalities is indicated:

1. The DESIGNATION_LIST contains an entry specifying a user from which the message path does not have the status CU.
2. There is no space available in the DESCRIPTION_AREA for another RECEIVE_TABLE entry.

The functional logic of the RECEIVE procedure is:

1. Validate the DESIGNATION_LIST, if any.
2. Search the DESTINATION_Q for a satisfying message.
3. If the message is found and AREA is given, move the message to the given area and delete it from the DESTINATION_Q.
4. If the message is found and AREA is not given, delete from the DESTINATION_Q the last previously accessed message of the same designation that has not been specifically RETAINed.
5. If the message is found and the ECB is given, POST it with a pointer to the message.
6. If no message is found, update the RECEIVE_TABLE.

<u>High-priority Message Handling</u>

A very useful application of the attention-routine facility is the receiving of such high-priority messages that normal message-processing activity should be interrupted in their favor. The call

```
SIMAR(IMAR,[DESIGNATION_LIST],[RETAIN],RESULT)
```

specifies the entry point of an Incoming Message Attention Routine (IMAR) to be entered whenever a message having one of the designations in the given DESIGNATION_LIST arrives in the program's DESTINATION_Q.

A call on SIMAR results in the updating of an IMAR_TABLE, which is a list of entries of the form (DESIGNATION$_t$,IMAR$_t$). The updating involves comparing each entry D_s in the given DESIGNATION_LIST with each DESIGNATION D_t in the IMAR_TABLE. The possible outcomes of this comparison are:

Condition	Action
$D_s \neq D_t$ for all D_t, IMAR $\neq 0$	(D_s,IMAR) appended to the IMAR table
$D_s \neq D_t$ for all D_t, IMAR $= 0$	None
$D_s = D_t$ for some D_t, IMAR $\neq 0$	(D_t,IMAR$_t$) replaced by (D_s,IMAR)
$D_s = D_t$ for some D_t, IMAR $= 0$	(D_t,IMAR$_t$) deleted from the IMAR_TABLE

A zero for either component of a DESIGNATION is interpreted as "any SOURCE" or "any TYPE," assumed to match the corresponding component of any other DESIGNATION. If the DESIGNATION_LIST parameter is omitted, a single DESIGNATION (0,0) is assumed. Thus, the entire IMAR_TABLE is either replaced by a single entry (0,IMAR) if IMAR is not equal to zero, or the entire table is deleted if IMAR equals zero.

A program's IMAR procedure is passed one parameter, MESSAGE_POINTER, which, when the IMAR is entered, points to the given message in the DESTINATION_Q.

If RETAIN is specified in the SIMAR call, the message is considered accessed when the IMAR is entered. A RETAINed message will not satisfy a subsequent RECEIVE request, and will be retained in the DESTINATION_Q until specifically released (see the RELEASE call, below). If RETAIN is not specified, the message is not considered accessed. The message may satisfy a subsequent RECEIVE call; it may be replaced by a subsequent message with the same TYPE or be deleted, unless specifically retained, if the DESTINATION_Q overflows.

Note that an arriving message that has been specifically requested via a RECEIVE call will satisfy that request without causing an IMAR to be scheduled.

RESULT can indicate one of the following abnormalities:

1. There is no space available in the DESCRIPTION_AREA for another IMAR_TABLE entry.
2. The DESIGNATION_LIST contains an entry specifying a SOURCE from which the message path does not have the status CU.

The SIMAR functional logic is:

1. Validate the DESIGNATION_LIST.
2. Search the IMAR_TABLE once for each entry in the given DESIGNATION_LIST, which effects either one or more deletions or replacements, or an addition to the IMAR_TABLE.
3. If the given IMAR has not been previously specified, declare it to OS/360 as a new attention routine.

Retaining Messages

RETAIN(MESSAGE_POINTER)

This call specifies that the message pointed to is to be retained in the DESTINATION_Q and not automatically deleted as a result of the arrival of later messages having the same designation. No result is returned. The functional logic is straightforward.

Deleting Messages

The call that specifies that one or more messages are to be dele from the DESTINATION_Q is:

RELEASE([DESIGNATION_LIST],[MESSAGE_POINTER])

Either one of the parameters must be given. If it is DESIGNATION_ LIST, all messages having any of the given designations are deleted. If it is MESSAGE_POINTER, the indicated message is deleted. No result is returned. The functional logic is straightforward.

SENDING

The call that sends a message having the given TYPE code to one or more terminals or program is:

```
SEND(TYPE,[DESTINATION_LIST,[ECBLIST]],
  [SECTION_LIST,[MOVE]],[ECB],RESULT)
```

DESTINATION_LIST is a list of USER_NUMBERs of VMH users to which the message is to be routed. If it is omitted, the message is routed to all users to which a CU message path exists.

ECBLIST is a list of ECBs--one for every specified user--to be POSTed when the message has been delivered to the corresponding program or transmitted to the corresponding terminal. This is the SENT event mentioned in Section IV.

SECTION_LIST is a linked list of entries, each entry containing the address and length of a section of the message, as well as an "ignore" flag. All sections for which the ignore flag is not set are concatenated to form the body of the message. The header, containing the TYPE code, USER_NUMBER of the source, and the length of the body, is constructed by VMH. If the SECTION_LIST parameter is omitted, the message consists of a header only.

The MOVE option means that the body of the message is to be moved into the program's MESSAGE_AREA prior to routing. This is a way of using the MESSAGE_AREA as an output buffer while the message is being transmitted to terminals. If this parameter is omitted, the message is transmitted or copied directly from its location in the program.

ECB, which may appear if ECBLIST does not, specifies an ECB that will be POSTed when delivery and/or transmission of the message to all destinations has been completed.

RESULT can indicate one of the following abnormalities:

1. The message was not routed to one or more of the indicated destinations because of improper message path status or DESTINATION_Q overflow conditions (corresponding entries in the DESTINATION_LIST, if it is given, are flagged).
2. The SECTION_LIST is constructed improperly.
3. Space is not available in the DESTINATION_AREA or the MESSAGE_AREA to store a SOURCE_Q element or otherwise process the message.

The functional logic includes:

1. Validating the DESTINATION_LIST, if specified, and determining the number of program and/or terminal destinations.
2. Validating the SECTION_LIST, if any, and determining the length of the message body.
3. Allocating space in the DESCRIPTION_AREA for a SOURCE_Q element and the necessary OUT_Q elements, and constructing the SOURCE_Q and OUT_Q elements.
4. If MOVE was specified, allocating space and moving the message into the MESSAGE_AREA.
5. If there are terminal destinations, constructing a channel output program in the DESCRIPTION_AREA, to be used by the OUTPUTTER.
6. POSTing the SOURCE event.

The SEND procedure is complicated by the necessity of being able to "unwind"--by returning all space allocated in the DESTINATION_AREA and MESSAGE_AREA--if SEND is unable to proceed because of lack of space or invalid specifications.

PARAMETER VALIDATION AND ABNORMAL TERMINATION OF PROGRAMS

Much of the programming of VMH service routines is concerned with the validation of lists and parameters provided by the calling program. As a vital link in access to a shared computer, VMH must protect itself and its users from errors or abnormal terminations caused by invalid data, particularly addresses or indices stored in the DESCRIPTION_AREA.

VMH also assists the debugging process by catching programming errors. Debugging a communication program, especially an asynchronous one, can be difficult. OS/360 provides almost no information as to the cause of a program check that occurs within an SVC routine--a fact that makes new SVCs quite difficult to debug. In addition, by detecting invalid data, VMH protects itself from the accusations of frustrated programmers ("I know everything was okay before I called SIMAR!").

What action should be taken when an apparent error is detected in a user program? The answer is not obvious. The policy adopted in the early versions of VMH was to notify programs of all detected abnormalities via return codes from the service calls, with a special "error attention routine" for those errors so serious that the program could not continue as a VMH user without some kind of programmed recovery. This approach was perhaps somewhat naive, because programmed recovery from programming errors is almost a contradiction in terms. Experience showed that programmers would neither provide error routines nor, in most cases, would they provide tests for more than one or two of the usual six or eight return codes.

Now the policy is to terminate the offending program as soon as an apparent programming error is detected, giving a unique termination code for every error detected in every service routine. (Under OS/360, a programmer can declare a special termination exit routine to receive control just before termination, which can instead attempt recovery.)

Following is a list of ten of the common termination-causing error conditions, out of the approximately 50 termination error conditions that the system recognizes.

1. The owner of a shared DESCRIPTION_AREA and MESSAGE_AREA has terminated.
2. An invalid address appears as a parameter or in a linked list.
3. A service call has been issued prior to the CONNECT call.
4. A DESIGNATION_LIST or DESTINATION_LIST contains too many entries (more than 256).
5. An invalid USER_NUMBER was given.
6. Invalid D_AREA_LIMITS or M_AREA_LIMITS were given.
7. An invalid MESSAGE_POINTER was given.
8. An invalid VGS message type was specified.
9. An invalid length was specified in a SECTION_LIST entry.
10. A SIMAR call was issued from an IMAR, which specified that the IMAR itself was to be deleted as an attention routine.

VI. THE TERMINAL INTERFACE

PHYSICAL CONTROL FACILITIES

VMH fulfills the role of message switcher in a symmetric way (described in Section III), with the same conventions for each VMH user, whether the user is a program or a terminal. VMH services are offered to programs as subroutine calls, but no such natural interface exists for VGS terminals.

Before describing this interface, it is necessary to discuss both the control facilities at a terminal and the messages that they generate. The common input device for every VGS terminal is a character keyboard with control box. Input messages generated via the keyboard consist of either a single character, or one or two lines of text. The control box has three buttons: CONNECT/CALL_VOS, PROGRAM_INTERRUPT, and DISCONNECT. VOS is the acronym for the Video Operating System.

The first use of the CONNECT/CALL_VOS button generates a LOGON message (discussed below), and every subsequent use before DISCONNECTing generates a VOS_ATTENTION message. The PROGRAM_INTERRUPT message is intended for use by the application program--for instance, to cause a working process to be interrupted by another. The others are assigned as "executive" messages, to be directed to the terminal interface in the service machine.

The I_DISTRIBUTOR checks the TYPE of every message coming from the VGS system. If it is one of the three executive types--LOGON, VOS_ ATTENTION, or DISCONNECT--it is not routed to user programs, but is delivered to VMH itself.

THE TERMINAL AGENT

Explicit communication control facilities are provided to the terminal by means of:

1. A shared reentrant program, called the TERMINAL_AGENT, which acts on behalf of the terminal in making CONNECT, DISCONNECT, and MESSAGE_PATH requests. There is one Terminal Agent task for each active terminal.

2. A special terminal mode, called "control mode," in which the person at the terminal can communicate directly with VMH via the TERMINAL_AGENT.

In normal mode, the terminal appears as a terminal to the message-handling procedures of VMH; but in control mode it appears to MESSAGE_PATH as a user program, able to request and to be notified of path status changes in the same way. The paragraphs below describe how this is achieved.

Log-on Processing

If the TYPE is LOGON, the message is delivered to a fifth VMH subtask, the LOGON_PROCESSOR, which owns a DESCRIPTION_AREA and MESSAGE AREA and accesses messages exactly as any other program-type user does. The LOGON_PROCESSOR is very simple: it does nothing with the message, but creates a task (via the OS/360 ATTACH SVC) as an incarnation of the TERMINAL_AGENT procedure and passes it a pointer to the message. (Hereafter, we use the term TERMINAL_AGENT to denote such a task, rather than denoting the procedure itself.) The TERMINAL_AGENTs, one for each logged-on terminal, are subtasks of the VMH job step, with dispatching priority just under that of the set of "VMH subtasks," the asynchronous procedures described in Section IV.

The TERMINAL_AGENT's first action is to CONNECT as a VMH user, identifying its role via a special option in the CONNECT call. The CONNECT procedure identifies this user as a TERMINAL_AGENT, or terminal-type user, in the USER_DESCRIPTION. The terminal's INTRINSIC_ID (as defined in Section III) is a unique terminal identification, obtained by CONNECT from the LOGON message and supplied by the VGS system in the header of every input message from that terminal. CONNECT also turns on a CONTROL_MODE indicator associated with this VMH terminal-type user, and deletes the LOGON message. The new TERMINAL_AGENT shares a DESCRIPTION_AREA and MESSAGE_AREA with the LOGON_PROCESSOR.

As far as VMH is concerned, the LOGON message is now processed, but the TERMINAL_AGENT has the remaining responsibility of completing the log-on sequence with the person at the terminal by obtaining his accounting information.

Control Mode

If the message TYPE is found by the I_DISTRIBUTOR to be VOS_ATTENTION, the message is delivered to the TERMINAL_AGENT for that terminal. Upon receipt of the message, the TERMINAL_AGENT immediately establishes the control mode for the terminal. It also opens a message path from the (terminal-type) user to itself--that is, TERMINAL_AGENT sets the message path status to CU. When the TERMINAL_AGENT sends a message over this path, VMH understands that the message is to be routed to the terminal instead. Note that a terminal and its TERMINAL_AGENT are regarded by VMH as the same VMH user.

When a terminal is in control mode, all communication between that terminal and other VMH users is suspended, and communication takes place only with its TERMINAL_AGENT.

When in control mode, the following conditions hold:

1. Every message path from another user that has the status CU is blocked, i.e., its status is set to CB. This is accomplished via a MESSAGE_PATH call, so that the other user is notified.
2. All messages from the terminal are delivered to its TERMINAL_AGENT.
3. All messages from programs to this terminal are delivered to its TERMINAL_AGENT also.

COMMANDS

Pushing the CALL_VOS button at the terminal, which puts the terminal in control mode, makes available a set of commands to inspect and to change the status of message paths involving the terminal. In addition, several auxiliary commands are available. Each command is typed on the keyboard and consists of a one-line message containing a keyword, followed by one or more operands. (The line is displayed, as it is typed, by the VGS system on the terminal's video display; it may be edited before being sent.)

Naming the Terminal

After completing the LOGON (by entering a line consisting of accounting information), the terminal may be named by the command

```
MYNAME <name>.
```

Naming is necessary only when communicating with terminals or with programs that are selective with respect to message sources and destinations. It is unnecessary if communication is to take place only with multi-access subsystems or services that accept message paths with terminals regardless of name, because the terminal initiates the path to the subsystem and VMH supplies the terminal's USER_NUMBER, when notifying the subsystem of the path status change. Naming may be done at any time after LOGON.

One of the following responses to the MYNAME command is displayed:

```
TERMINAL NAMED <name>.
```

The terminal now has the given NAME.

```
<name> BEING USED.  TRY ANOTHER NAME.
```

The NAME entered was being used by another VMH user.

```
TERMINAL ALREADY NAMED <name>.
```

A terminal may be named only once between LOGON and DISCONNECT, and NAME may not be changed.

Creating Message Paths

The command

```
CREATE <direction><name>
```

initiates a message path to and/or from another VMH user, or completes a path or paths already initiated by another VMH user. The parameter

<direction> is one of the character graphics "←", "→", or "↔", denoting "the path to", "the path from", or "the paths to and from," respectively. One of the following responses is displayed:

AWAITING <name>. PGM INT ENDS WAIT.

No VMH user has the given NAME, so the terminal must wait for notification that a user has CONNECTed under that NAME. Alternatively, the CREATE command may be cancelled by pushing the PROGRAM_INTERRUPT button. (The keyboard cannot be used because it is locked by the VGS system after every keyboard message and can be unlocked only by a message from a program. The TERMINAL_AGENT is in a waiting state.)

<direction> INITIATED WITH <name>.

The indicated paths were successfully initiated or completed.

STATUS ALREADY EXISTS WITH <name>.

This is self-explanatory.

INCORRECT STATUS FOR DUAL REQUEST.

<direction> was "↔" but the indicated message path did not have a dual status (see The Message Path Scheme, Section III).

Deleting Message Paths

The command

DELETE <direction><name>

changes the status of a message path (or paths) to and/or from another VMH user to NL. One of the following responses is displayed.

NO USER WITH NAME <name>.

The terminal has no paths with a user by that NAME.

<direction> DELETED WITH <name>.

The deletion was successful.

STATUS ALREADY EXISTS WITH <name>.

The indicated message paths have already been deleted.

INCORRECT STATUS FOR DUAL REQUEST.

<direction> was "↔", but the indicated message path did not have a dual status (see The Message Path Scheme, Section III).

<u>Establishing Communication Mode (USE)</u>

After the desired message paths have been completed, the terminal can be taken out of control mode and put into communication with any subset of programs with which complete paths exist, by the command

USE <name><name>...<name>.

The list may include only names of programs from which complete paths exist.

The effect of the command is that

1. The status of the path to the terminal from each of the named users (assuming the path is already complete) is set to CU; i.e., the path is unblocked.

2. Subsequent messages from the terminal are routed to every user to which the path has the status CU.

3. Subsequent messages from programs to this terminal are transmitted to it.

Any of the following responses indicates an error:

NO COMPLETE PATH FROM <name>.

The path from the indicated program is not complete.

TERMINAL HAS NO PATHS WITH <name>.

This is self-explanatory.

PATHS and EVENTS Displays

While his terminal is in control mode, the person using it may command his TERMINAL_AGENT to show him one of two displays: the Path Status Display, showing the users with whom his terminal has message paths, and the Event Record, which shows the status changes that these message paths have undergone. When the person pushes the CALL_VOS button, returning his terminal to control mode from any other mode, his TERMINAL_AGENT automatically chooses the Path Status Display for him.

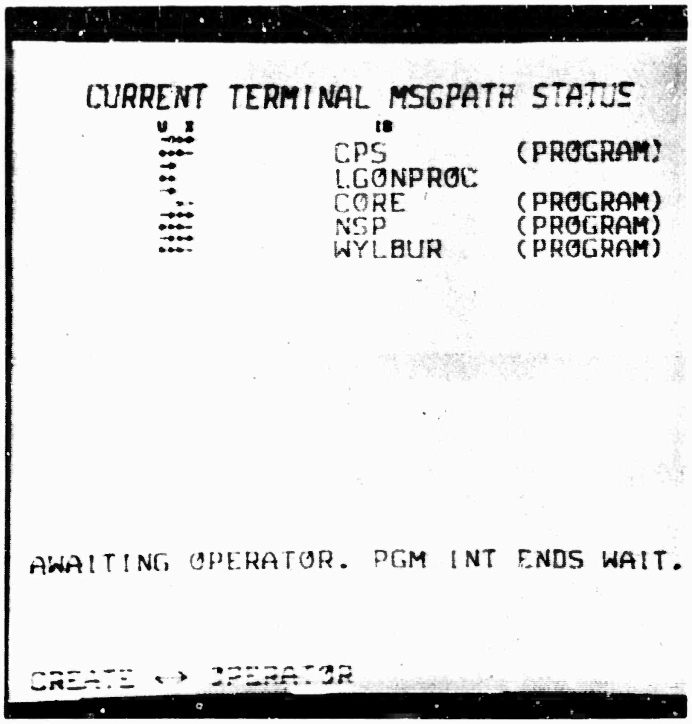
Whenever a person is entering a keyboard line or pair of lines, the VGS system displays them on the screen. When the terminal is in control mode, these lines appear at the bottom of the screen. The rest of the screen is used by the TERMINAL_AGENT to maintain and update the Path Status Display or the Event Record. The Path Status Display remains until the person either selects the Event Record to replace it, or enters an error-free USE command.

The Paths Display (CURRENT TERMINAL MSGPATH STATUS) shows the current status of all non-null paths between the terminal and other users. Figure 6a is an example. The arrows beneath "U" denote the status of the terminal's ("your") link pair with the other user ("X"); the arrows beneath "X" denote the status of the other user's link pair with the terminal (see The Message Path Scheme, Section III, for an explanation of link pairs). The notation consists of an arrow for each link, with a barred arrow indicating a blocked path.

The Terminal Event Record, illustrated in Fig. 6b, provides a limited history of status changes to all non-null paths involving this terminal. The display contains a fixed number of entries, each formatted as:

<time><direction><action><actor><name>.

<time>	The time of day, to the nearest ten seconds, at which the event occurred.
<direction>	One of the character graphics "→", "←", or "↔", indicating "the path to", "the path from", or "the paths to and from," respectively.



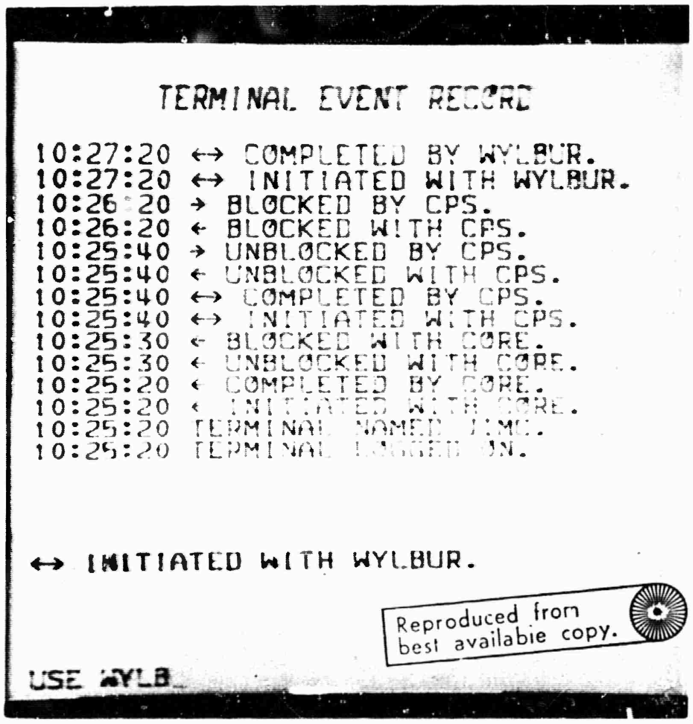
```
CURRENT TERMINAL MSGPATH STATUS
                 CPS          (PROGRAM)
                 LGONPROC
                 CORE         (PROGRAM)
                 NSP          (PROGRAM)
                 WYLBUR       (PROGRAM)



AWAITING OPERATOR. PGM INT ENDS WAIT.


CREATE ↔ OPERATOR
```

Fig. 6a — Path status display



```
        TERMINAL EVENT RECORD

10:27:20 ↔ COMPLETED BY WYLBUR.
10:27:20 ↔ INITIATED WITH WYLBUR.
10:26:20 → BLOCKED BY CPS.
10:26:20 ← BLOCKED WITH CPS.
10:25:40 → UNBLOCKED BY CPS.
10:25:40 ← UNBLOCKED WITH CPS.
10:25:40 ↔ COMPLETED BY CPS.
10:25:40 ↔ INITIATED WITH CPS.
10:25:30 ← BLOCKED WITH CORE.
10:25:30 ← UNBLOCKED WITH CORE.
10:25:20 ← COMPLETED BY CORE.
10:25:20 ← INITIATED WITH CORE.
10:25:20 TERMINAL NAMED JIMC.
10:25:20 TERMINAL LOGGED ON.


↔ INITIATED WITH WYLBUR.


USE WYLB_
```

Fig. 6b — Event record

<action>	One of INITIATED, COMPLETED, BLOCKED, UNBLOCKED, or DELETED; they are self-explanatory.
<actor>	Either WITH or BY; BY if the other party took the initiative, WITH if the user did.
<name>	The name of the other user.

As a new event occurs, it replaces the oldest previous event in the display. By a feature not yet implemented on currently used terminals, the TERMINAL_AGENT will also produce an audio tone at the terminal to notify the person that a path status has changed.

VII. IMPLEMENTATION

SUB-SURFACE DETAILS

In addition to the primary functions of VMH, a significant amount of programming effort was devoted to details of implementation, such as:

1. Accounting and event logging.
2. Communication with the system operator.
3. Input and output channel programming, with associated error checking and handling.
4. Initiation, initialization, and normal or abnormal termination of VMH.
5. Interception of normal or abnormal terminations of user programs.
6. Automatic updating of all users' RECEIVE_TABLEs and IMAR_TABLEs.
7. Validity checking and diagnostic routines.
8. "Compensating" routines for space allocation and subtask control.
9. Special features of the SEND service, included to give programs more control over the real-time activity of interactive picture manipulation: (a) notification of the start and completion of message transmission to a terminal, via an ECB or an OMAR (Outgoing Message Attention Routine), and (b) the ability to delete or replace a message in the OUT_Q before transmission to a terminal has begun.
10. Handling exceptional conditions that might arise from: (a) failure of an input or output channel; (b) an overload of output messages in the VOS system or in VMH itself; (c) inability to schedule an attention routine because its interrupt queue is full; and (d) inability to accept a user because the VMH storage region is fully occupied.

The last two overload conditions might arise because of the fixed storage regions allocated by OS/360 and the basic lack of relocatability of data and programs in System/360. To avoid the time-consuming space allocation routines of OS/360, VMH allocates its space via its own

Message and Data Space Allocation system, described below. The VMH DESCRIPTION_AREA consists of blocks of 2048 bytes because that is the minimum unit of storage protected in System/360.

A message must be moved into the program's region because of the basic lack of segmentability in the System/360 addressing structure, which requires programs to operate in contiguous regions of primary storage. A special subtask controller (described in Subtask Control, below) is necessitated by the lack of generality in the task synchronization primitives of OS/360, as mentioned in Event Queues, Message Queues, and Sequencing, Section IV.

MESSAGE AND DATA SPACE ALLOCATION

We describe here the method by which VMH dynamically allocates and releases space for various data elements in the DESCRIPTION_AREA and for messages in the MESSAGE_AREA of a program. The two areas are handled identically. The method used is a modification of the "buddy system."(7)

Free blocks occur in sizes of 2^K bytes, where the integer K ranges from K_1 through K_2 (in VMH, $K_1 = 4$ and $K_2 = 11$). Free blocks of each size are linked together into a unidirectional list, and an indexed FREE_SPACE_TABLE contains K_2-K_1+1 list headers, the K*th* header consisting of a "combined" flag and a link to the first free block, if any, of size 2^K.

A call for a block of space is handled as follows:

1. The amount of the request, A, is translated into an integer K such that $2^{K-1} < A \leq 2^K$.

2. If the K*th* list is non-empty, the first block in that list is unlinked and assigned to the caller. Nearly all requests for space are satisfied in this way.

3. Otherwise, headers K+1, K+2, ..., K_2, are examined until a non-empty list K' is found. The first block on this list is unlinked and its two halves put on list K'-1. This procedure is repeated for lists K'-1, K'-2, ..., K+1. There are now two blocks of the required size; after one of them has been assigned to the caller, there remains one block each on lists K, K+1, ..., K'-1.

4. If no larger block is found, headers K-1, K-2, ..., K_1 are examined until a list K'' is found that is both non-empty and "uncombined"--which means that it may contain one or more pairs of "buddies" (space-contiguous blocks, the first of which starts at offset $j \cdot 2^{K''+1}$ bytes from the beginning of the area for some integer $j \geq 0$). This list is searched for buddies; any found are combined and put on list K''+1. This procedure is repeated for lists K''+1, K''+2, .. , K-1 until a block of the required size has been obtained. If no buddies are found, the search for non-empty and uncombined lists continues from list K''-1.

5. If no free block is obtained from steps (2), (3), or (4), the request cannot be satisfied from the currently allocated area. If the size of the area has not reached the maximum specified in the AREA_ LIMITS parameter of the CONNECT call (see Control Services, Section V), the OS/360 GETMAIN SVC is called, on behalf of the program, to expand the area by a certain amount, 2^m (m is a VMH system parameter). If the request is successful, the block obtained is put on list K'=m, and step (3) is repeated.

When a block of space is returned by the user, it is entered as the first block on the list of free blocks of that size, but no combining is done. By combining blocks only when necessary, the space-return operation becomes trivial. Even better, the free-space lists become "adapted" so that the block-sizes most frequently requested (and returned) are the ones most probably available. In VMH, this has proved extremely effective; our statistics indicate that at least 99.8 percent of all space requests are satisfied by step (2), making the get-space operation in these cases almost as simple as the return-space.

The obvious problem with a technique that allocates fixed-size blocks is "internal fragmentation"--the waste of 2^K-A bytes to satisfy a request for A bytes. In VMH, this is not a serious problem because many of the elements stored in the DESCRIPTION_AREA, as well as several of the VGS messages, are of lengths equal to or just under a power of two.

SUBTASK CONTROL

The VMH CONTROLLER (mentioned on page 21) is a very simple, very specialized process supervisor, managing the communication between subtasks of the VMH job step and the transfer of control among them. It consists of the EVENT and WAITV routines and the DISPATCHER. The primary goal in designing the CONTROLLER was to allow subtasks to generate one or more instances of work for others, even though these other subtasks had not completed processing previous instances--i.e., to provide for event buffering. Secondly, subtasks had to be able to synchronize on non-local events, signalled by tasks not under control of the VMH CONTROLLER (e.g., those signalled by OS/360), as well as on local events, signalled by other VMH subtasks.

On the other hand, there is no need to create tasks or event-types dynamically, as they are all predefined. Furthermore, all events of a given type are processed by the same subtask. Finally, although a subtask may wait for different event-types at different stages of its processing, it need wait for only one event at a time.

Thus each event-type is identified by an index, both in the EVENT and WAITV primitives. Non-local event-types have event buffers of size one (the ECBs themselves); buffers of larger but finite size are associated with local event-types.

An EVENT call, signalling a local event, causes an "event data pointer" (EDP) provided with the call to be stored in the associated event buffer if no subtask is waiting for this event. If the buffer is full, the calling task is put into a special wait state and control is given to the ready subtask of highest priority (if any). (Buffer sizes are chosen to minimize the occurrence of this condition; if it should occur, a system diagnostic message will be printed.) If a subtask is awaiting this event, the EDP is stored in the ECB associated with the wait and the "posted" bit in that ECB is set to one. This bit determines whether a subtask is "ready" or not.

Except as noted above, control is returned to the issuing subtask. We decided that it was not worthwhile to check for a higher priority subtask, that might have been readied by this event, and pass control to it, because the time between WAITVs in any subtask is short.

When a WAITV call is issued to await a non-local event-type, control goes directly to the DISPATCHER routine of the CONTROLLER. If the WAITV is for a local event-type, the associated event buffer is checked. If it is non-empty, the oldest EDP is removed from the event buffer and placed in the ECB provided with the call, and the "posted" bit is set in this ECB. (If the buffer is full, the subtask that filled it is taken out of its special wait state.) Control then goes to the DISPATCHER routine.

The DISPATCHER searches the list of ECBs associated with the VMH subtasks. (This list is ordered according to the subtask priorities: (1) INPUTTER, (2) OUTPUTTER, (3) I_DISTRIBUTOR, (4) O_DISTRIBUTOR, and (5) LOGON_PROCESSOR.) Control is given to the first task for which the associated "posted" bit is set. This bit is also set by OS/360 when the POST SVC is called to signal a non-local event. If no posted ECB is found, the list of ECBs is used as the parameter of an OS/360 WAIT SVC, which is then issued by the VMH CONTROLLER to await any activity.

Timing measurements found the CONTROLLER, i.e., the EVENT and WAITV routines and the DISPATCHER, to consume less than 3 percent of total message handling time, averaging about 45 instructions per message (see Size and Performance, below).

THE DATA BASE

The VMH data base consists of (1) a single VMH_NUCLEUS, (2) a single USER_TABLE, and (3) a USER_DESCRIPTION for each user.

The VMH_NUCLEUS is the core of the data base. It resides in the nucleus of OS/360, and contains (1) the entry point addresses of all utility routines used by VMH services and subtasks, (2) pointers to the rest of the data base, and (3) VMH-wide parameters, counts, and flags, as shown on the following list. Addressability to the VMH_NUCLEUS is obtained by VMH via an SVC that is private to VMH. It is initialized by the VMH initialization program.

ELEMENTS OF VMH_NUCLEUS

Entry points to OS/360 supervisor services

POST

GETMAIN

FREEMAIN

Entry points to VMH-wide utility procedures

SPACE_MANAGER

CREATE_ATTN_ROUTINE_REQUEST_BLOCK

DELETE_ATTN_ROUTINE_REQUEST_BLOCK

SCHEDULE_ATTN_ROUTINE

TERMINATE_USER_ABNORMALLY

NULLIFY_ALL_PATHS

Pointers to other VMH data

USER_TABLE

VGS_MESSAGE_TYPE_TABLE

VMH_STATISTICS_TABLE

TERMINAL_USER_NUMBER_TABLE

Other data

MAX_VALID_USER_NUMBER

MAX_CURRENT_USER_NUMBER

FIRST_AVAILABLE_USER_NUMBER

LOGON_PROC_USER_NUMBER

MAX_VALID_TERMINAL_ID

INPUT_CHANNEL_DOWN_FLAG

OUTPUT_CHANNEL_DOWN_FLAG

The USER_TABLE contains pointers to the USER_DESCRIPTION of all users, plus flags indicating the type of user and the user's status. Space for the USER_TABLE is allocated in the VMH region. It is indexed by USER_NUMBER which, for programs, is stored in the TCB (see p. 11). The USER_NUMBER of a terminal is obtained from a table, also in the VMH region, that is indexed by TERMINAL_ID.

All other information describing a VMH user is stored in the DESCRIPTION_AREA of that user. The basic element is the USER_DESCRIPTION, whose entries are listed below. It contains pointers to the queues and tables associated with messages or with table elements governing message sending and reception. Some of the latter are shown on the following page.

CONTENTS OF THE USER_DESCRIPTION

NAME
INTRINSIC_ID
GET_USER_NUMBER_NAME_WAITED_FOR
PTR_TO_GET_USER_NUMBER_ECB
ENTRY_TO_MPAR
ENTRY_TO_OVFLAR
OVERFLOW_CONDITION_FLAG
OVERFLOW_OPTION
PTR_TO_PATH_STATUS_TABLE
PTR_TO_RECEIVE_TABLE
PTR_TO_IMAR_TABLE
PTR_TO_OMAR_TABLE
PTR_TO_SOURCE_Q
PTR_TO_DESTINATION_Q
PTR_TO_FREE_SPACE_TABLE_D_AREA
PTR_TO_FREE_SPACE_TABLE_M_AREA

PER-USER MESSAGE INFORMATION

Each RECEIVE_TABLE entry

AREA

ECB

DESIGNATION_LIST

Each IMAR_TABLE entry

ENTRY_TO_IMAR

DESIGNATION_LIST

RETAIN_OPTION

Each DESTINATION_Q entry

PTR_TO_MESSAGE_IN_M_AREA

PTR_TO_FREE_SPACE_HEADER_FOR_MESSAGE_BLOCK

Each SOURCE_Q entry

Message Header:

USER_NUMBER_OF_SENDER
TYPE
LENGTH

PTR_TO_DESTINATION_LIST

PTR_TO_ECB_LIST_OR_ECB

PTR_TO_SECTION_LIST_OR_CHANNEL_PROGRAM

MOVE_OPTION

ENTRY_TO_OMAR

NOTIFICATION_OPTION

PTR_TO_TABLE_OF_M_SPACE_OCCUPIED

PTR_TO_USER_DESCRIPTION

Each OUT_Q entry

PTR_TO_SOURCE_Q

PTR_TO_ASSOC_ECB

USER_NUMBER_OF_DESTINATION

VMH SIZE AND PERFORMANCE

How Large Is VMH?

It is difficult to state the size of VMH in figures commensurate with the description in this report, because the essential VMH--the facilities described in Sections III--VI--would, if programmed afresh, be perhaps only 60 percent as large as the VMH system as it was actually implemented. The "overhead" code provides features not covered in this report and areas that have been mentioned here but not fully described.

As implemented, the VMH program includes approximately 11,000 OS/360 Assembler Language source statements (not counting comments, listing controls, or macro-generated statements). In addition, there are about 3,000 macro-definition statements, primarily for data-base element descriptions and calling sequences for the service SVCs. In operation, VMH occupies nearly 40,000 bytes of storage, not counting the data bases of user programs.

We estimate that VMH required about four man-years of labor for design, coding, checkout, and final documentation. Of this, checkout required an inordinate amount of time, perhaps 35 to 40 percent, using the most primitive methods: sitting at the machine console or poring over core dumps to track down the causes of some of the most elusive malfunctions of the "non-recurring" character with which implementers of real-time systems are familiar.

Performance

The statistics in Table 4 were obtained by monitoring normal operation on a 360/40 during several periods of early applications. The total time required to handle a single message having a single destination was found to average about 7.5 msec on input (INPUTTER + I_DISTRIBUTOR, including DELIVER + RECEIVE processing times) and about 9.5 msec on output (SEND + O_DISTRIBUTOR + OUTPUTTER processing times), for a total "turnaround" time apparently averaging less than 17 msec, and a throughput of roughly 120 messages per second. (The lower figures in Table 4 are apparently more indicative than the upper ones.)

Table 4

PROCEDURE TIMING DATA
(msec on a 360/40)

INPUTTER (per message)	2.2 - 3.2
OUTPUTTER (per message)	3.5 - 4.1
I_DISTRIBUTOR (single destination)	2.9 - 3.9
O_DISTRIBUTOR (single destination)	1.2 - 1.6
CONTROLLER (per dispatch)	0.5 - 1.0
RECEIVE (per call)	1.1 - 2.6
SEND (per call)	3.4 - 4.3
Total turnaround time	17 sec
Throughput	120 messages/sec

We observed that a program in the 360/40 can barely maintain a real-time display of a Rand Tablet that generates position data at intervals of 1/60 second. On a 360/65, maintaining the same display uses about 14 to 20 percent of the computer's central processor time, for a maximum throughput on that machine of about 600 to 850 messages per second.

Evaluation

As with most large systems, a number of VMH design features are open to criticism and possible improvement. The whole scheme for communication control should be reexamined. It does what it was designed to do, but there is some question whether all its features are needed. The value of a Message Path Attention Routine and the ability to specify both sources and destinations of messages seems slight, in our experience at Rand. In most current applications, when a program is notified of an initiated message path, it automatically completes the path, and most current programs use a nondirected RECEIVE and a directed SEND. The message path status is ordinarily not used to determine message routing, but must be checked to validate those sources and destinations that are explicitly specified.

Also, message path control seems unnecessarily inconvenient to the person at a terminal who simply wishes to communicate with a single program. After logging on, he must CREATE the paths to and from the program and then enter the USE command, when he would rather just enter the program name. A command system should be as transparent as possible to the person at a terminal: he should not be made conscious of the control details but should be free to concentrate on his problem.

A feature that has never been regarded as satisfactory, from the first, is the manner of handling DESTINATION_Q overflows. This will never be a problem for those applications in which messages from and to another user are synchronized. In the asynchronous case, however, overflow can be avoided only if the destination program is allocated time "on the average" to process messages at the rate they arrive. The only general solutions are an always-expandable message area, a guaranteed allocation of sufficient processing time, or the ability to reduce the rate or shut off the supply of messages--each of which may be impossible or inappropriate in given contexts. An initial approach to the second solution, a guaranteed processing rate, has been implemented in VOS with a "Consistent Timeslice Scheduler." So far, the only application for asynchronous processing has been use of the Rand Tablet for graphical input. When Tablet data are discarded because of overflow, the result is deterioration in the quality or reliability of response.

With respect to internal design, there are at least three deficiencies:

First, because most of the message-handling part of VMH operates as a separate job step under OS/360, it is difficult to account for message-handling on a per-user basis. Even though the activity of each procedure could be measured and charged to some program or terminal, it is not always clear on whose behalf the activity is being performed; the accounting itself would add to message-processing time. Therefore, message handling is regarded as "system overhead."

Second, the DELIVER procedure executes as a shared subroutine of the I_DISTRIBUTOR and O_DISTRIBUTOR. In case a message is to be distributed to many programs, this could mean a slight performance degradation because one VMH subtask may not be interrupted by another.

In view of the time consumed by DELIVER per destination, it might have been better to implement it as a separate subtask with its own event queue. This alternative was not considered during the design and has not been evaluated.

Finally, it may seem strange that a message from a program to N terminals results in N entries in the OUT_Q and N transmissions instead of one, particularly because they are all sent to the VGS control computer for further routing. This duplication was made necessary by the design of VGS. The use of program-to-terminal multiplexing in this way may never be frequent enough to warrant a change.

On the positive side, the message-path control scheme is a general and flexible method of treating communication control in a symmetric system of many users. The strategy of m[illegible] this control on a level distinct from the level at which messages are actually exchanged has two major advantages: (1) It allows users to be more completely and currently informed about the activities of other users. (2) The scheme is easily implemented, with an extremely simple interface to the actual message-handling procedures; hence, it is easily modified and adapted to new requirements simply by adding new control requests and status values.

In summary, VMH is an unusual system, providing interprogram as well as program/terminal communications, with highly similar treatment of the different types of users. The Video Graphics System, of which VMH is an essential part, has been in almost continuous operation at Rand since early 1971, supporting 32 low-cost graphic terminals.

VMH has been occupied largely with supporting interactive terminal applications. IBM's CPS (Conversational Programming System)(8) has been adapted for VGS terminals and is in service daily. Many applications involve interactive graphics, using the Rand Tablet as an input device. One of these, BIOMOD, is a facility for simulating models of continuous systems, for example those involving chemical reactions where the chemical equations may be entered in natural form.(9) Another, Data Analysis System, is a powerful tool for the statistical analysis of data bases. A third, MIND (Management of Information through Natural Discourse) is a system for natural language processing which includes a variety of functions, such as translation and information retrieval.(10-14) Some applications have used VMH as an interprocess communication facility between tasks of the same job or between jobs themselves (across region boundaries)--which, as mentioned earlier, is not otherwise possible under OS/360. The VGS system is the means of access at Rand to the facilities of the nationwide ARPA Computing Network.(5) We still anticipate its use in multiterminal cooperative problem solving or gaming, for which VMH is a particularly appropriate message exchange.

REFERENCES

1. Bell, T. E., *Modeling the Video Graphics System: Procedure and Model Description,* The Rand Corporation, R-519-PR, December 1970.
2. Uncapher, K. W., *The Rand Video Graphic System--An Approach to a General User-Computer Graphic Communication System,* The Rand Corporation, R-753-ARPA, April 1971.
3. Davis, M., and T. O. Ellis, *The RAND Tablet: A Man-Machine Graphical Communication Device,* The Rand Corporation, RM-4122-ARPA, August 1964.
4. Brown, G. D., and C. H. Bush, *The Integrated Graphics System for the IBM 2250,* The Rand Corporation, RM-5531-ARPA, October 1968. (The general system description applies to the present Video Graphics System although the output device mentioned in the title is different.)
5. Ellis, T. O., E. F. Harslem, J. F. Heafner, and K. Uncapher, *ARPA Network Series: I. Introduction to the ARPA Network at Rand and to the Rand Video Graphics System,* The Rand Corporation, R-664-ARPA, September 1971.
6. *IBM Operating System/360, MVT Guide,* Form GC28-6720-1, International Business Machines Corp., Poughkeepsie, N.Y., September 1, 1970.
7. Knuth, Donald E., *The Art of Computer Programming: Vol. 1, Fundamental Algorithms,* Addison-Wesley Publishing Co., Reading, Mass., 1968.
8. *IBM Conversational Programming System (CPS): System Programmer's Guide,* Form GH20-0757-0, International Business Machines Corp., White Plains, N.Y., January 1970.
9. Groner, G. F., R. L. Clark, R. A. Berman, and E. C. DeLand, *BIOMOD: An Interactive Computer Graphics System for Modeling,* The Rand Corporation, R-617-NIH, July 1971.
10. Shapiro, Stuart Charles, *The MIND System: A Data Structure for Semantic Information Processing,* The Rand Corporation, R-837-PR, August 1971.
11. Kaplan, Ronald M., *The MIND System: A Grammar-Rule Language,* The Rand Corporation, RM-6265/1-PR, April 1970.
12. Kay, Martin, and Gary R. Martins, *The MIND System: The Morphological-Analysis Program,* The Rand Corporation, RM-6265/2-PR, April 1970.
13. Kay, Martin, and Stanley Y. W. Su, *The MIND System: The Structure of the Semantic File,* The Rand Corporation, RM-6265/3-PR, June 1970.
14. Bisbey, Richard L., *A Tree Grapher for the Linguist,* The Rand Corporation, P-4730, November 1971.